11. Hardware interfacing

Een controller bordje heeft aan de zijkanten een rij pinnen. Dat zijn de GP of GPIO-pinnen (General Purpose Input/Output). In dit hoofdstuk gaan we na hoe deze pinnen gebruikt worden voor het aansluiten van allerlei elektronica, zoals LED's, schakelaars, sensoren, relais, … . De meeste pinnen hebben meerdere functies.

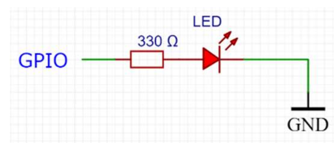
Digitale in- en output pinnen

Alle GPIO's kunnen gebruikt worden als digitale input/output-pin. Een digitale ingang kan de stand van een schakelaar of informatie van een sensor uitlezen. Met een digitale uitgang stuur je actuatoren aan zoals LEDs en motoren.

De controller gebruikt signaalniveaus van 3,3 volt. Een digitale output heeft waarde 1 (3,3 volt) of 0 (0 volt). Een digitale input leest 3,3 volt als 1 en 0 volt als 0.

Sluit nooit spanningen hoger dan 3.3V aan op een input pin, de controller zal beschadigd worden.

Met een LED kan je het signaalniveau van een output-pin zichtbaar maken. Sluit de LED met een weerstand (330 tot 560 Ω) aan tussen de outputpin en een massapunt. De weerstand zorgt voor dat de stroom door de led niet te groot wordt

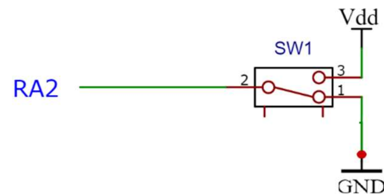


Figuur 44: LED, aansluitschema

De Raspberry Pi Pico heeft een LED aan poort 25. Die is aan als poort 25 hoog is. De Pimoroni Pico RP2040 heeft een driekleurenled aan poorten 18, 19 en 20, één kleur per poort. De LED's zijn aan als de poorten laag zijn. Sommige bordjes hebben een neopixel LED, de besturing daarvan is een stuk ingewikkelder. (zie later). Er bestaan ongelooflijk veel verschillende bordjes. De meeste hebben een LED aan boord maar er geen enkele standaard voor de aansluiting van die LED aan een GPIO. Kijk in de technische documentatie van het bordje of ga na of er op het bordje informatie over de LED te vinden is.

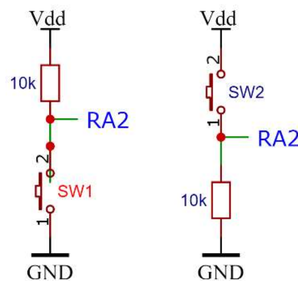
In de voorbeelden in dit en volgende hoofdstukken gebruiken we veel GPIO-poorten. Het was niet mogelijk om eenduidige regels te maken voor de keuze van die poorten. Ga na welke poorten in jouw systeem bruikbaar zijn en pas de toepassing aan.

Een digitale inputpoort kan je o.a. gebruiken om de stand van een schakelaar uit te lezen. Een open inputpoort (niets aangesloten) heeft een onbepaald niveau, het lezen van die poort geeft niet te voorspellen resultaten. We kunnen met een wisselschakelaar gebruiken om de poort te verbinden met de voeding Vcc of de massa.



Figuur 45; logische niveaus met een wisselschakelaar

In de linkerschakeling hieronder heeft RA2 niveau 0 V bij een gesloten schakelaar. Bij een open schakelaar "trekt" de weerstand, de **pull-up-weerstand** het niveau naar Vdd. De GPIO heeft altijd een duidelijk ingangsniveau. De rechter schakeling is omgekeerd: daar "trekt" de **pull-down weerstand** het niveau naar 0 V bij een open schakelaar.



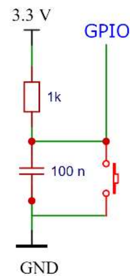
Figuur 46: pull-up en pull-down weerstanden

De meeste poorten hebben een ingebouwde pull-up- en een pull-down-weerstand. Je activeert die bij de configuratie van de poort. De externe weerstand kan je dan weglaten.

Contactdender van ingangspoorten

Een vervelende eigenschap van mechanische schakelaars is dat ze bij het sluiten enkele keren stuiteren tegen het contact. Dit kan enkele milliseconden duren, dat is lang t.o.v. de snelheid van de controller. Tijdens het denderen zal de schakelaar enkele tientallen keren contact maken en verbreken. Sommige schakelaars hebben er meer last van dan andere, maar meestal moet je iets aan doen. Er bestaan sofTware oplossingen voor, bij voorbeeld door enige tijd niet te reageren op een

verandering. Een eenvoudige hardware oplossing maak je met een condensator over de contacten van de schakelaar. Samen met de pull-up of pull-down weerstand geeft dit een effectieve onderdrukking van de contactdender.



Figuur 47: onderdrukken van contactdender

Het schema hierboven toont de contactdenderonderdrukking bij een drukknop met pull-up weerstand. De waarde op de GPIO is 1 bij geopende schakelaar en 0 bij indrukken. Het ontdenderen van een drukknop met pull-down-weerstand is volledig analoog.

GPIO-poorten en MicroPython

Klasse Pin van module machine controleert de GPIO's. De algemene vorm is

Machine.Pin(id, mode, pull, value, drive, alt)

De eerste twee parameters zijn verplicht.

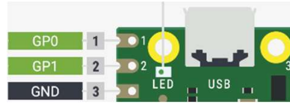
- id: is het nummer van de GPIO
- mode bepaalt of de pin in- of output is en heeft waarde PIN.IN of PIN.OUT
- pull bepaalt de pull-up weerstand. Waarden zijn PULL_UP, PULL_DOWN en PULL.None.
- value geeft het logische niveau van de pin.

Voor digitale poorten gebruiken we methodes **Pin.value()** :

- Pin.value() geeft de waarde van een inputpin
- Pin.value(n) zet waarde n op een output.

Voorbeeld:

Sluit een LED aan tussen GPIO1 en GND, een schakelaar tussen GPIO0 en GND. Met de schakelaar bedienen we de LED. Hieronder zie je de plaats van de poorten op een RP Pi Pico..



Figuur 48: aansluitgegevens voor led-switch.py (RP Pi Pico)

```
#-----------------------------#
#  led-switch.py              #
#                             #
#  19 februari 2021           #
#  Dirk Ghysels               #
#-----------------------------#
from machine import Pin
led = Pin(1, Pin.OUT)
switch = Pin(0, Pin.IN, Pin.PULL_UP)
while True:
    if switch.value()==1:
        led.value(1)
    else:
        led.value(0)
```

Variabelen led en switch zijn vertegenwoordigers van klasse Pin. led definiëren we als een outputpin, switch als een input met pull-up weerstand. Als de input een 1 leest, dan brandt de led.

Inputpin en interrupts

In vorig voorbeeld controleren we de toestand van een schakelaar door telkens weer de toestand op te vragen. Dat is niet altijd de beste manier. We weten niet wanneer de gebruiker de schakelaar gaat gebruiken. Het is beter als het programma een normaal verloop kent, maar dat de schakelaar het programma even onderbreekt als hij wordt ingedrukt. Dat doen we met een **interrupt** of een **IRQ** (interrupt request). Alleen een input kan een IRQ activeren.

Voorbeeld:

We zetten een drukknop van GP0 naar Vcc en we activeren de pull-down weerstand. Zodra iemand die indrukt, moet de LED aan GP1 branden. *Verander in het programma de definitie van led en button als je andere GPIO's gebruikt.*

```
from machine import Pin
led = Pin(1, Pin.OUT)
led.value(0)
button = Pin(0, Pin.IN, Pin.PULL_DOWN)

def button_handler(pin):
    led.value(1)

button.irq (trigger=Pin.IRQ_RISING, handler=button_handler)
```

De laatste regel activeert de IRQ, gekoppeld aan inputpin button.

- Trigger geeft aan hoe de IRQ geactiveerd wordt:
    - IRQ_RISING: door een opgaande flank van een puls op de pin
    - IRQ_FALLING: door een neergaande flank van een puls op de pin
- Handler geeft aan welke functie de IRQ oproept. Deze krijgt als parameter de pin van de button mee.

De touch-ingangen van de ESP32

Een ESP32 heeft een aantal touch-ingangen. Dat zijn ingangen voor een sensor die reageert op aanraken. De sensor is gewoon een metalen plaatje dat met een draadje verbonden is met een GPIO. Alleen GPIO's 0, 2, 4, 12, 13 14, 15, 27, 32 en 33 kan je gebruiken. Voor de aansturing een touchpad gebruiken we klasse Touchpad uit module machine. De uitlezing geeft een lage waarde bij uitlezing en een hoge zonder. De exacte waarde hangt o.a. af van vorm van het plaatje, de lengte van de verbindingsdraad en de manier van aanraken. In het voorbeeldprogramma is een grenswaarde 200 gebruikt, die kan je experimenteel bepalen. Aanraken laat een LED branden. De sensor is gemaakt met een metalen plaatje en een stukje draad.

```
#-----------------------------#
#  touch.py                   #
#                             #
#  18 december 2021           #
#  Dirk Ghysels               #
#-----------------------------#
from machine import TouchPad, Pin
from time import sleep
led = Pin(25, Pin.OUT)
tp = TouchPad(Pin(4))
while True:
    tpr = tp.read()
    led.value(1)
    if (tpr>200):
        led.value(0)
    sleep(1)
```



Figuur 49: touch-pad sensor

analoge output

Een echte analoge output – een signaal met een instelbare gelijkspanning- kunnen we niet maken met de controller. Computers werken digitaal en een uitgang kan alleen maar spanning 0V of Vcc leveren. We kunnen wel de gemiddelde waarde van een signaal instelbaar tussen 0V en Vcc maken door het signaal een blokvorm te maken (afwisselend HIGH en LOW) met vaste frekwentie en de verhouding tussen de lengte van HIGH en LOW te variëren. Dergelijke signalen noemt men **PWM**, **pulse width modulation** of **puls breedte modulatie**. Een toepassing is het dimmen van een LED. Als de frekwentie hoog genoeg is, dan ziet men geen flikkeringen, wel een variatie in helderheid door het aanpassen van de HIGH-LOW verhouding. De frekwentie hangt af van de toepassing: enkele keren per minuut voor een elektrisch fornuis, tot 100 Hz voor een LED, enkele kHz voor motorsturing en honderden kHz voor schakelende voedingen.

Een eerste methode is gebaseerd op programma blink.py, waarmee we een LED laten knipperen. De frequentie wordt hoger, we willen geen geflikker zien.

```
#------------------------------#
#  led-dimmer.py               #
#                              #
#  19 februari 2021            #
#  Dirk Ghysels                #
#------------------------------#
from machine import Pin
from time import sleep
led = Pin(1, Pin.OUT)
switch = Pin(0, Pin.IN, Pin.PULL_UP)
while True:
    led.value(1)
    sleep(0.002)
    led.value(0)
    sleep(0.008)
```

Het programma maakt een blokgolf. Een HIGH duurt 2 ms, een LOW 8 ms. Eén aan/uit cylus duur 10 ms, de frequentie is 100 Hz. De LED is 25% van de tijd aan, De LED is gedimd.

De RP2040 en de ESP32 hebben modules waarmee je tot 16 PWM-signalen tegelijkertijd kunt produceren. Dat is onmogelijk met de methode die we hierboven gezien hebben. Je kunt elke GPIO gebruiken voor PWM. Een groot voorbeeld is dat een PWM-module onafhankelijk van de processor werkt. De gebruiker geeft een opdracht een PWM-signaal te genereren. Dat signaal blijft actief en ondertussen kan het programma andere dingen doen. De RP2040 kan PWM-signalen genereren van 7 Hz tot 125 MHz, de PM's van de ESP32 gaan van 1 Hz tot 40 MHz.

MicroPython gebruikt klasse PWM uit library machine voor PWM signalen.

Klasse PWM voor de ESP32:

```
from machine import Pin, PWM

pwm0 = PWM(Pin(0))      # maak een PWM object op GPIO0
pwm0.freq()             # geef de frekwentie
pwm0.freq(1000)         # stel frekwentie in: 1000 Hz
pwm0.duty_u16()         # geef de duty cycle, in bereik 0-65535
pwm0.duty_u16(200)      # stel duty cycle in, bereik 0-65535
pwm0.deinit()           # schakel PWM uit
```

Klasse PWM voor de RP2040:

```
from machine import Pin, PWM

pwm0 = PWM(Pin(0))      # maak een PWM object op GPIO0
pwm0.freq()             # geef de frekwentie
pwm0.freq(1000)         # stel frekwentie in: 1000 Hz
pwm0.duty ()            # geef de duty cycle, in bereik 0-1023
pwm0.duty_ (200)        # stel duty cycle in, bereik 0-65535
pwm0.duty_u16()         # geef de duty cycle, in bereik 0-65535
pwm0.duty_u16(200)      # stel duty cycle in, bereik 0-65535
pwm0.duty_ns(250_000)   # stel pulse width in tussen
     0 en 1_000_000_000/freq, (hier: 25%)
pwm0.duty_ns()          # get current pulse width in ns
pwm0.deinit()              # turn off PWM on the pin
```

Voorbeeld:

Het programma hieronder genereert een PWM-signaal op poort 1 om een LED te dimmen. De helderheid van de LED stijgt van 0 tot maximum, en daalt dan verder tot 0.

```
#------------------------------#
#  led-PWM.py                  #
#                              #
#  20 februari 2021            #
#  Dirk Ghysels                #
#------------------------------#
from machine import Pin, PWM
from time import sleep
pwm = PWM(Pin(1))
pwm.freq(1000)
while True:
    for dc in range(0, 65536, 5000):
        pwm.duty_u16(dc)
        sleep(0.2)
    for dc in range(65535, -1, -5000):
            pwm.duty_u16(dc)
            sleep(0.2)
```

De analoog-digitaal convertor

De analoog-digitaal convertor of ADC vertaalt een analoge waarde op een ADC-ingang naar een digitale waarde. Het analoge signaal wordt vergeleken met een referentiespanning en wordt dan omgezet naar een binair getal.

De ADC van de ESP32

De standaard referentiespanning is 1 volt, de resolutie van het resultaat is 12 bit. Met 12 bits kan je 4096 getallen voorstellen (2^{12}= 4096), het resultaat van de omzetting is een getal tussen 0 en 4095.

> spanning = 0 volt → resultaat = 0,
> spanning = 1 volt → resultaat 4095
> spanning = X volt → resultaat R = 4095.X

Bij een gegeven resultaat R van de ADC, is de ingangsspanning X = R./ 4095

Een spanning groter dan 1 volt geeft uitlezing 4095.

De ADC van de RP2040

De standaard referentiespanning is 3,3 volt, de resolutie van het resultaat is 12 bit. Het resultaat wordt omgezet naar een getal tussen 0 en 65535.

> spanning = 0 volt → resultaat = 0,
> spanning = 3,3 volt resultaat 654536
> spanning = X volt resultaat R = 654536.X / 3,3

Bij een gegeven resultaat R van de ADC, is de ingangsspanning X = R.3,3 / 65535

Een spanning groter dan 3,3 volt kan de controller beschadigen !!

De tabel toont op welke pinnen we de DAC's terugvinden.

ESP32		**RP2040**	
pin	**ADC**	**pin**	**ADC**
GPIO 26	ADC0	GPIO 26	ADC0
GPIO 27	ADC1	GPIO 27	ADC1
GPIO 28	ADC2	GPIO 28	ADC2
		GPIO 29	ADC3

We gebruiken de ADC met klasse ADC uit library machine. De constructor wordt

> adc = machine.ADC(*n*)

n is het nummer van de ADC.

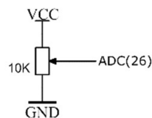
Voor het uitlezen van de DAC gebruiken we één van de read-functies:

```
from machine import ADC

adc = ADC(pin)        # creëert een ADC-object
val = adc.read_u16()  # leest de waarde in bereik 0-65535
val = adc.read_uv()   # read een analoge waardi in microvolts
```

Voorbeeld:

We gebruiken de ADC op GPIO 26 om een spanning te meten. Als spanningsbron gebruiken we een potentiometer tussen Vcc en GND. De loper verbinden we met GPIO 26. We meten twee maal per seconde. Een te snelle opeenvolging van metingen geeft foute resultaten. Wacht even tussen twee metingen. Het programma staat hieronder. Verander de stand van de potentiometer, je ziet direct nieuwe resultaten.



Figuur 50: meten van spanningen

```
#------------------------------#
#  ADC.py                      #
#                              #
#  11 maart 2021               #
#  Dirk Ghysels                #
#------------------------------#
import machine, utime
adc = machine.ADC(26)
while True:
    R = adc.read_u16()
    spanning = R*3.3/65535
    print ("spanning = %f volt" %spanning)
    utime.sleep(.5)
```

De interne temperatuursensor van de RP2040

De RP2040 heeft een ADC die verbonden is met een interne temperatuursensor. Volgens het datasheet is de temperatuur te berekenen uit de ADC-spanning met formule:

$$T = 27 – (ADC_{spanning} – 0.706)/0.001721$$

Dit levert volgend programma:

```
#------------------------------#
#  ADC-temp.py                 #
#                              #
```

```
#  11 maart 2021              #
#  Dirk Ghysels               #
#-----------------------------#
import machine, utime
adc = machine.ADC(4)
while True:
    R = adc.read_u16()
    S = R*3.3/65535
    T = 27 - (S-0.706)/0.001721
    print ("Temperatuur = %f °C" %T)
    utime.sleep(.5)
```

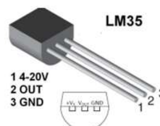
De sensor bevindt zich in de controller en meet dus de temperatuur van de controller. Als de lucht vrij kan circuleren boven de controller zal de temperatuur van de controller weinig verschillen van de omgevingslucht. Bij zware berekeningen of zware belasting van de GPIO's, zal de controller warmte genereren. De temperatuurmeting zal minder nauwkeurig worden. De controller van compacte bordjes zit onderaan het printje. De lucht kan minder circuleren en de meting is ook onnauwkeuriger.

Toepassing: temperatuur meten met de LM35

De **LM35**, 36 en 37 zijn en de compatibele TMP35, 36 en 37 zijn temperatuursensoren met analoge uitgang. We kunnen hun uitgang inlezen met een analoge ingang.

	LM35/TMP35	LM36/TMP36	LM37/TMP37
Output	10 mV/°C	500 mV + 10 mV/°C	20 mV/°C
Temperatuur	10°C - 125°C	-40°C - 125°C	5°C - 125°C
Nauwkeurigheid bij 25 °C	± 1°C		
Nauwkeurigheid over volledig bereik	± 2°C		
Voeding	2.7 V – 5.5 V		

De LM35 zet temperatuur om in spanning: 10 mV/ °C. De uitgang, pin 2, sluiten we aan op een ADC-poort. Pin 3 is GND die verbinden we met GND van het Pico-bordje en pin 1, de voeding aan de 5V aansluiting. Negatieve temperaturen kan je er niet mee meten. We willen geen spanning hoger dan 3,3 volt aan een ADC-ingang, dat beperkt de meting tot 330 °C. Voor huiskamergebruik is dat zeker geen probleem.



Figuur 51: LM35, aansluitschema

Uit de waarde van de ADC kan de temperatuur berekend worden:

T = 0 °C → V_{out} = 0V → ADC = 0

T = 330 °C → V_{out} = 3,3 V → ADC = 65535

T = t °C → ADC = (t x 65535)/330

Of t = (330.ADC)/65535

In het programma is de LM35 aangesloten op ADC0. Het resultaat is afgerond op één cijfer na de komma, dat is voldoende, de sensor heeft een nauwkeurigheid van 1 °C.

```
#------------------------------#
#  LM35.py                     #
#                              #
#  12 maart 2021               #
#  Dirk Ghysels                #
#------------------------------#
import machine, utime
adc = machine.ADC(0)
while True:
    R = adc.read_u16()
    temp = R*330/65535
    print ("temperatuur = %.1f °C" %temp)
    utime.sleep(.5)
```